Lecture Notes in Machine Learning – Chapter 4: Version space learning

Zdravko Markov

February 17, 2004

Let us consider an example. We shall use an *attribute-value* language for both the examples and the hypotheses $L = \{[A, B], A \in T_1, B \in T_2\}$. T_1 and T_2 are taxonomic trees of attribute values. Let's consider the taxonomies of colors (T_1) and planar geometric shapes (T_2), defined by the relation *son*.

```
Taxonomy of Colors:                  Taxonomy of Shapes:

son(primary_color,any_color).        son(polygon,any_shape).
son(composite_color,any_color).      son(oval,any_shape).
son(red,primary_color).              son(triangle,polygon).
son(blue,primary_color).             son(quadrangle,polygon).
son(green,primary_color).            son(rectangle,quadrangle).
son(orange,composite_color).         son(square,quadrangle).
son(pink,composite_color).           son(trapezoid,quadrangle).
son(yellow,composite_color).         son(circle,oval).
son(grey,composite_color).           son(ellipse,oval).
```

Using the hierarchically ordered attribute values in taxonomies we can define the derivability relation ($\rightarrow$) by the *cover* relation ($\geq$), as follows: $[A_1, B_1] \geq [A_2, B_2]$, if A_2 is a successor of A_1 in T_1, and B_2 is a successor of B_1 in T_2. For example, $[red, polygon] \geq [red, triangle]$, and $[any\_color, any\_shape]$ covers all possible examples expressed in the language L.

Let $P \in L, Q \in L$, and $P \geq Q$. Then P is a *generalization* of Q, and Q is a *specialization* of P.

Let $E^+ = \{E_1^+, E_2^+\}, E_1^+ = [red, square], E_2^+ = [blue, rectangle]$,
$E^- = [orange, triangle]$, and $B = \oslash$.

Then the problem is to find such a hypothesis H, that $H \geq E_1^+$, $H \geq E_2^+$, i.e. H is a *generalization* of E^+.

Clearly there are a number of such generalizations, i.e. we have a *hypothesis space*
$S_H = \{[primary\_color, quadrangle], [primary\_color, polygon]$, ...,
$[any\_color, any\_shape]\}$.

However not all hypotheses from S_H satisfy the consistency requirement, ($H \not\geq E^-$), i.e. some of them are *overgeneralized*. So, the elements $H \in S$, such that $H \geq E^-$, have to be excluded, i.e they have to be *specialized*, so that not to cover any more the negative example. Thus we obtain a set of *correct* (consistent with the examples) hypotheses, which is called *version space*, $VS = \{[primary\_color, quadrangle], [any\_color, quadrangle]\}$.

Now we can add the obtained hypotheses to the background knowledge and further process other positive and negative examples. Learning systems which process a sequence of examples one at a time and at each step maintain a consistent hypotheses are called *incremental learning systems*. Clearly the basic task of these systems is to search through the version space. As

we have shown above this search can be directed in two ways – *specific to general* and *general to specific*.

1 Search strategies in version space

To solve the induction problem the version space have to be searched through in order to find the best hypothesis. The simplest algorithm for this search could be the generate-and-test algorithm, where the generator produces all generalizations of the positive examples and the tester filters out those of them which cover the negative examples. Since the version space could be very large such an algorithm is obviously unsuitable. Hence the version space has to be structured and some directed search strategies have to be applied.

1.1 Specific to general search

This search strategy maintains a set S (a part of the version space) of *maximally specific generalizations*. The aim here is to avoid overgeneralization. A hypothesis H is maximally specific if it covers all positive examples, none of the negative examples, and for any other hypothesis H' that covers the positive examples, $H' \geq H$. The algorithm is the following:

Begin

Initialize S to the first positive example

Initialize N to all negative examples seen so far

For each positive example E^+ do begin

Replace every $H \in S$, such that $H \not\geq E^+$, with all its generalizations that cover E^+

Delete from S all hypotheses that cover other hypotheses in S

Delete from S all hypotheses that cover any element from N

End

For every negative example E^- do begin

Delete all members of S that cover E^-

Add E^- to N

End

End

1.2 General to specific search

This strategy maintains a set G (a part of the version space) of *maximally general hypotheses*. A hypothesis H is maximally general if it covers none of the negative examples, and for any other hypothesis H' that covers no negative examples, $H \geq H'$. The algorithm is the following:

Begin

Initialize G to the most general concept in the version space

Initialize P to all positive examples seen so far

For each negative example E^- do begin

Replace every $H \in G$, such that $H \geq E^-$, with all its specializations that do not cover E^-

Delete from G all hypotheses more specific (covered by) other hypotheses in G

Delete from G all hypotheses that fail to cover some example from P

End

For every positive example E^+ do begin

Delete all members of G that fail to cover E^+

Add E^+ to P

End

End

2 Candidate Elimination Algorithm

The algorithms shown above generate a number of plausible hypotheses. Actually the sets S and G can be seen as boundary sets defining all hypotheses in the version space. This is expressed by the *boundary set theorem* [1], which says that for every element H from the version space there exist $H' \in S$ and $H'' \in G$, such that $H \geq H'$ and $H'' \geq H$. In other words the boundary sets S and G allows us to generate every possible hypothesis by generalization and specialization of their elements, i.e. every element in the version space can be found along the generalization/specialization links between elements of G and S. This suggests an algorithm combining the two search strategies of the version space, called *candidate elimination algorithm* [2].

The candidate elimination algorithm uses bi-directional search of the version space. It can be easily obtained by putting together the algorithms from section 2.1 and 2.2 and replacing the following items from them:

1. Replace "Delete from S all hypotheses that cover any element from N" with "Delete from S any hypothesis not more specific than some hypothesis in G"

2. Replace "Delete from G all hypotheses that fail to cover some example from P" with "Delete from G any hypothesis more specific than some hypothesis in S"

These alterations are possible since each one of them implies what it alters. Thus collecting all positive and negative examples in the sets P and N becomes unnecessary. Clearly this makes the bi-directional algorithm more efficient. Furthermore using the boundary sets two stopping conditions can be imposed:

1. If $G = S$ and both are singletons, then stop. The algorithm has found a single hypothesis consistent with the examples.

2. If G or S becomes empty then stop. Indicate that there is no hypothesis that covers all positive and none of the negative examples.

3 Experiment Generation, Interactive Learning

The standard definition of the inductive learning problem assumes that the training examples age given by an independent agent and the learner has no control over them. In many cases, however, it is possible to select an example and then to acquire information about its classification. Learning systems exploring this strategy are called *interactive learning systems*. Such a system use an agent (called oracle) which provides the classification of any example

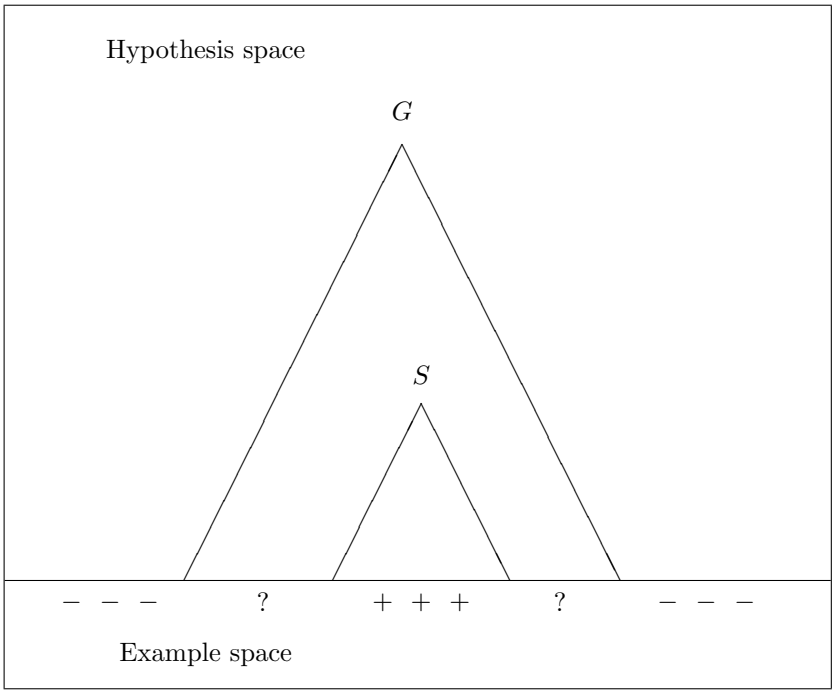

Figure 1: Graphical representation of version space. Question marsk denote the areas from where new examples are chosen

the systems asks for. Clearly the basic problem here is to ask in such a way that the number of further questions is minimal.

A common strategy in such situations is to select an example which halves the number of hypotheses, i.e. one that satisfies one halve of the hypotheses and does not satisfy the other halve.

Within the framework of the version space algorithm the halving strategy would be to find an example that does not belong to the current version space (otherwise its classification is known - it has to be positive) and to check it against all other hypotheses outside the version space. Clearly this could be very costly. Therefore a simple strategy is the following (this is actually and interactive version of the candidate elimination algorithm):

1. Ask for the first positive example
2. Calculate S and G using the candidate elimination algorithm
3. Find E, such that $G \geq E, \forall s \in S, E \not\geq s$ (E is not in the version space).
4. Ask about the classification of E
5. Go to 2

The exit from this loop is through the stopping conditions of the candidate elimination algorithm (item 2). A graphical illustration of the experiment generation algorithm is shown in Figure 1

References

[1] M. Genesereth and N. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

[2] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.